B-Tracker: Improving load balancing and efficiency in distributed P2P trackers

Hecht, Fabio V ; Bocek, Thomas ; Stiller, Burkhard

Abstract: Trackers are used in peer-to-peer (P2P) networks for provider discovery, that is, mapping resources to potential providers. Centralized trackers, e.g., as in the original BitTorrent protocol, do not benefit from P2P properties, such as no single point of failure, scalability, and load balancing. Decentralized mechanisms have thus been proposed, based on distributed hash tables (DHTs) and gossiping, such as BitTorrent's Peer Exchange (PEX). While DHT-based trackers suffer from load balancing problems, gossip-based ones cannot deliver new mappings quickly. This paper presents B-Tracker, a fully-distributed, pull-based tracker. B-Tracker extends DHT functionality by distributing the tracker load among all providers in a swarm. Bloom filters are used to avoid redundant mappings to be transmitted. This results in the important properties of load balancing and scalability, while adding the ability for peers to fetch new mappings instantly. B-Tracker shows, through simulations, improved load balancing and better efficiency when compared to pure DHTs and PEX.

DOI: https://doi.org/10.1109/P2P.2011.6038749

University of Zurich
Department of Informatics

Fabio Hecht
Thomas Bocek
Burkhard Stiller

B-Tracker: improving load balancing and efficiency in distributed P2P trackers

April 2011

TECHNICAL REPORT – No. 2011.IFI-2011.0003

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

F. Hecht
T. Bocek
B. Stiller:
Technical Report No. 2011.IFI-2011.0003, April 2011
Communication Systems
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

University of Zurich
Department of Informatics

Fabio Hecht
Thomas Bocek
Burkhard Stiller

B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers

April 2011

TECHNICAL REPORT – No. 2011.IFI-2011.0003

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

F. Hecht, T. Bocek, D. Hausheer, B. Stiller:
Technical Report No. 2011.IFI-2011.0003, April 2011
Communication Systems Group (CSG), Institute of Informatics (IFI), University of Zurich
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch

Chapter 1

Introduction

Trackers are important building blocks used in peer-to-peer (P2P) systems for provider discovery — mapping *resources*, such as files or video segments, to *providers*, that is, peers that announce the ability to provide them. In the simplest case, *e.g.* the original BitTorrent protocol [5], the tracker follows a client/server (C/S) approach.

C/S-based trackers, however, do not fully benefit from P2P properties, such as no single point of failure, scalability, load balancing, and the lack of a central authority. Therefore, different types of distributed trackers have been deployed. Distributed hash tables (DHTs) are natural candidates to be used as distributed trackers [11, 6], since their main functionality is mapping keys (content) into values (providers). Another way of designing a distributed tracker is using a gossip protocol, such as Peer Exchange (PEX) [3, 1], to allow peers to spread information about potential providers.

This paper introduces B-Tracker (Balanced Tracker), a fully-decentralized, pull-based tracker that improves both efficiency and load balancing. The main idea is that each provider becomes itself a tracker for the resources it provides. Algorithms are introduced for tracker discovery and updating information. In addition, a Bloom filter [4] is used to avoid peers discovering providers they already know.

B-Tracker may be used by any P2P application that uses a tracker to locate possible providers. These include file-sharing applications, such as BitTorrent. Delay-sensitive applications, *e.g.* live streaming, may have additional benefits by requesting peers when those are needed, since B-Tracker is pull-based. After obtaining a provider list, peers attempt to establish connections to each provider; if successful, these peers become *neighbors* and are enabled to exchange data.

Evaluations show that the proposed approach achieves both better efficiency and load balancing when compared to a pure DHT approaches and PEX.

The remainder of this paper is organized as follows. Section 2 presents related work. The suggested approach is described in Section 3. Evaluation details and results are presented in Section 4. Section 5 contains conclusions and future work.

Chapter 2

Related Work

Different types of distributed trackers have been proposed and deployed so the tracker can benefit from P2P advantages. They are divided into DHT approaches and gossiping.

Distributed hash tables (DHTs), such as KAD [11, 6], are able to map *keys* (*e.g.*, content) into *values* (*e.g.*, providers). The DHT functionality needs to be modified to allow several values to be added to a single key and to return a random subset of those values when queried. DHT-based trackers are pull-based, which allows a peer to retrieve a new set of providers as soon as and only as long as it is needed. The fact that a random subset is returned, regardless of which providers the requester already has already obtained, reduces its efficiency, since a large amount of traffic may be used to transfer useless providers. Another problem is load balancing. Since content popularity resembles a power-law distribution [8], and DHTs keep a constant number of replicas per key, the peers responsible for popular content have a much higher load than others.

Another way of designing a distributed tracker is using a gossip protocol, such as Peer Exchange (PEX) [7, 2, 3], which is implemented by several BitTorrent clients as an extension of the original protocol. Using PEX does not eliminate the need for a tracker, since every peer must still contact the tracker (C/S or DHT) in order to receive its first provider list. Though different implementations of PEX exist, their main idea is that peers keep their neighbors informed about their current neighbor set. This is generally done by periodically (*e.g.*, once a minute) sending messages containing sets of added and removed neighbors [1] to every neighbor. When new providers are needed, peers select those peers that appear least frequently as their neighbors' neighbors. The reason to choose the less popular ones is that those are probably newly arrived and need new neighbors. Load balancing is expected to be better in PEX, since every peer is responsible for sending regular update messages. But, since gossip protocols are push-based, they need to consider a trade-off on the frequency of gossip messages sent. If sent less frequently, the information is spread more slowly, which may be troublesome, especially for delay-sensitive applications, such as video streaming. If sent more frequently, efficiency decreases, as information will be more redundant.

Table 2.1 shows a comparison between the expected efficiency and load balancing properties of DHT, PEX and B-Tracker. Efficiency refers to the traffic generated per peer to

Table 2.1: Related Work Comparison

Approach	*Efficiency*	*Load Balancing*	*Push/pull*
DHT	-	-	Pull
PEX	-	+	Push
B-Tracker	+	+	Pull

spread the knowledge of providers, while load balancing refers to how well the traffic is distributed among the peers.

Chapter 3

B-Tracker Design

Though the terms *peer*, *tracker*, *provider*, and *neighbor* all refer to the a participant in the P2P system, the terminology defines more precisely the different roles that peers perform, even though every participant is expected to perform all roles in different situations. In short, a *peer* queries a *tracker* to obtain a list of *providers*, which are contacted directly and, if there is mutual interest, may become a *neighbor*, with which actual resource provisioning takes place. The basic functions a tracker offers to peers are *getProviders(resourceID)*, which returns a list of providers of the resource, and *addAsProvider(resourceID)*, which adds the sender of the message to the provider list at the trackers. Finally, *removeAsProvider(resourceID)* is called by a peer that is not anymore a provider for the given resource. It is assumed that, once a peer is provided with a resource, *e.g.* a file or a video stream, it becomes itself a provider of it.

3.1 Primary Trackers

B-Tracker uses a DHT structure for initial tracker discovery of a resource, since DHTs offer a scalable structure to store key-value mappings at well-known locations. Peers with *peerID* closest to the key (the *resourceID*) are responsible for storing the list of providers of the resource. These peers – termed *primary trackers* – are discovered in $O(\log n)$ steps, where n is the number of peers in the system. The DHT's original *put(key, value)* function is modified to allow multiple tuples (*peerID, IP address, TCP or UDP port* of providers) to be stored under a single key, and *get(key)* is adapted to return a random subset of these tuples.

The number of peers that are primary trackers for a resource is given by the primary tracker replication factor r_p. Since primary trackers are not necessarily providers for the resources they track, and to motivate peers to use secondary trackers, they have limited provider storage capacity, holding only up to c_p providers per resource.

3.2 Secondary Trackers

Once a peer has obtained a provider list from a primary tracker, subsequent tracker queries can be issued to any provider, since each of them is a *secondary tracker* for the resources they provide. The concept of secondary trackers improves scalability, since resources with more providers are able to distribute the load among more trackers, and fairness, because the load is shared by those peers interested in providing the resource.

An important parameter is n_p, the number of primary trackers that peers consult before requesting from a secondary tracker. While a low value decreases the load of primary trackers, it may return peers that have none or outdated information about secondary trackers. A high value reduces this risk, but increases the number of requests to primary trackers. Secondary trackers are not limited in storage capacity.

3.3 Improving Efficiency

In order not to suffer from the Coupon Collector Problem [10], and to avoid that each tracker keeps state about which providers were supplied to each request, a solution based on Bloom filters [4] is used. Queries to primary and secondary trackers include a Bloom filter with all already known providers. Trackers take the filter into consideration by returning a random subset of known providers for the specified resource, excluding providers that match the filter. While Bloom filters save bandwidth due to their fixed size, they may produce false positives. This means that an unknown provider may not be found. The probability of false positives can be adjusted to be low.

3.4 Updating Mechanism

The *addAsProvider(resourceID)* operation is used by peers, which have become providers for a certain resource. It can be issued to both primary and secondary trackers, but primary trackers accept only up to c_p providers per resource.

While primary trackers are a fixed set of peers found via the DHT logic, secondary trackers are potentially much more numerous. Different strategies can be used for choosing the subset of secondary trackers to send *addAsProvider(resourceID)* messages. An intuitive strategy, used by B-Tracker, is to choose a random subset of known secondary trackers.

The replication factor r_p determines on how many primary trackers this information is stored, while r_s refers to the number of replicas at secondary trackers. A higher r_s value increases the chance that a provider is found, while update and maintenance operations are more expensive.

3.5 Outdated Information

Tracker information tends to become outdated as peers fail, leave, or stop offering resources, without informing the responsible trackers. This is a problem for all centralized or distributed tracker approaches. Having the tracker verify all providers it holds is too large an effort due to the potentially large number of entries. B-Tracker assigns a time to live (TTL) to each resource-provider mapping stored. Providers need to update (via *addAsProvider(resourceID)*) their respective tracker entries before the TTL runs out.

Chapter 4

Evaluation

B-Tracker has been implemented and evaluated using simulations to show its properties, when compared to other distributed tracker approaches, namely PEX and pure DHT-based trackers. TomP2P [12] has been adapted to support DHT, PEX, and B-Tracker approaches.

The evaluation focuses on efficiency and load balancing. Load is defined in terms of upstream traffic, since it is a scarce resource in a P2P system. Efficiency is defined in terms of mean load per peer in the swarm, considering all tracker-related messages sent, so less load conveys better efficiency. Load balancing is defined as the standard deviation of load among all peers in the swarm, so less deviation determines a better balance.

The parameters used for the evaluation are as follows. The Bloom filter assumes a probability of false positives $p = 0.0073$ with a number of items $n = 100$, which result in a filter of size $m = 1024$ bits [4]. A fixed replication factor $r_p = 20$ is used for the DHT approach, as in the popular BitTorrent implementation [6]. B-Tracker uses $r_p = 2$ and $r_s = 18$ for replication, since they add up to 20, in order to be fairly comparable to the DHT approach. The number of primary trackers that peers consult before requesting from a secondary tracker $n_p = 0$, that is, they always query secondary trackers for providers first, resorting to primary trackers only if all queries to secondary trackers fail. Primary tracker storage capacity $c_p = 35$ providers per resource, since 35 is a common number of neighbors used by P2P applications. All results are averages from 100 runs.

4.1 Simulation Setup

A P2P system with 1000 peers was simulated as follows. At each run, a swarm is initially created with 50, 250, or 450 peers. Peers in the swarm are interested in obtaining a certain resource, *e.g.*, downloading a file. Each peer in the swarm obtains 35 providers from the DHT, which is a common size in BitTorrent. Measurement starts only after they have obtained those initial providers, in order to simulate a live swarm. The system, then, suffers from churn, which is defined as the percentage (10%, 20%, 30%, or 40%) of peers in the swarm that go offline, being immediately substituted by the same number of newly

created peers. All peers attempt in turn to have again 35 providers in total, exchanging messages according to the approach in place.

In the DHT approach, peers query always a random one of the 20 peers that are responsible for holding the provider list for the resource in question. The tracker always replies with a random subset of at most 35 providers.

In the PEX approach, peers exchange PEX messages containing a set with newly added neighbors and a set of disconnected neighbors. If, after exchanging PEX messages, a peer still does not have 35 providers, it queries the DHT to obtain them.

In the B-Tracker-NF – NF stands for *no Bloom filters* – approach, peers query one or more random secondary trackers, which in essence are providers obtained from the initial DHT call until they obtain at least 35 providers. If, after querying all known secondary trackers, a peer has not reached its goal, it queries a primary tracker to obtain them. The B-Tracker approach works like B-Tracker-NF, except that all requests contain a Bloom filter holding the currently known providers.

4.2 Evaluation: Efficiency

In a more efficient system, the knowledge of which are current providers is spread with less traffic generated per peer. Efficiency is, therefore, defined in terms of the average load per peer; load being defined as bytes sent per peer, on average. Figures 4.1, 4.2, and 4.3 show the average load per peer for swarm sizes 50, 250, and 450, respectively. Each value shown is an average of all runs, with error bars displaying the standard deviation.

B-Tracker achieves very good efficiency when compared to pure DHT and PEX approaches. A DHT approach is not efficient because each new peer and peers with less than 35 providers in the swarm need to query the DHT, which creates many routing messages to find the trackers. PEX is even less efficient, because it requires that peers send many unnecessary messages, informing neighbors about their new neighbors regardless of whether or not they need them. B-Tracker shows better efficiency than B-Tracker-NF due to the use of Bloom filters – though request messages are larger, since they contain the filter, the provider list returned by trackers contains only useful information.

The B-Tracker approach shows good scalability, since, for larger swarms, the mean load per peer increases only slightly. DHT and PEX experience a larger load increase from swarm size 50 to 250, though from 250 to 450 it increases only slightly. The difference between B-Tracker-NF and B-Tracker also increases with a larger swarm size.

Load also increases with more churn for all investigated approaches, since, with more churn, there are more newly created peers that look for providers, and more peers need to obtain more providers. PEX, however, has a higher load increase with higher churn when compared to the other approaches.

4.3 Evaluation: Load Balancing

Figures 4.4, 4.5, and 4.6 show the standard deviation in load among all peers in the swarm. Load balancing was calculated for each run and an average for all runs is displayed; error bars show, thus, the standard deviation for the different runs.

B-Tracker-NF and B-Tracker distribute load much better than DHT, due to the presence of secondary trackers. In a pure DHT approach, peers that are trackers become heavily loaded as swarm size increases, as seen in Figure 4.6. PEX shows improved load balancing, especially on larger swarms. B-Tracker-NF shows that using Bloom filters as proposed improves the load balancing only by a small amount. The fact that load balancing degrades on larger swarms on all approaches is explained by their use of the DHT at least for initial tracker discovery.

Churn has a negative influence on load balancing in all investigated approaches and swarm sizes. This is also due to the DHT being queried at least initially by all new peers. The difference of these load balancing steps, however, is small between 30% and 40% churn rates, suggesting that it increases at smaller steps.

4.4 Evaluation: Detailed View

Figures 4.7-4.18 show in more detail the peer load in the different scenarios studied and help understanding previously shown results of efficiency and load balancing. An average of the top 150 peers with highest load on the different runs is shown. Note that some peers that are not part of the swarm may be slighly loaded due to sending DHT routing messages.

It can be seen that the top 20 peers have a significantly higher load than the others, due to them being responsible for the holding the DHT values of the resource sought after. In PEX, load is much better distributed than on the pure DHT approach, but there is still a peak on the most loaded peers, since it is still needed to resort to the DHT frequently. This happens because PEX alone does not always spread provider information to all peers. B-Tracker further spreads load to more peers, and reduces mean load when compared to the other approaches, but higher churn still increases load unevenly among peers.

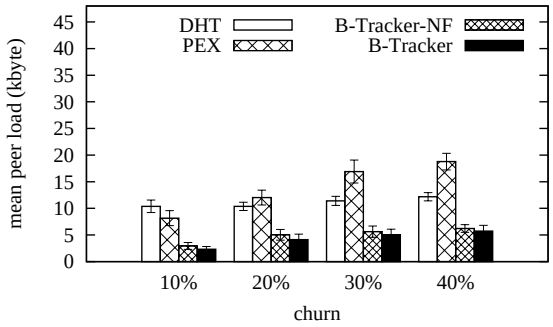

Figure 4.1: Efficiency, swarm size 50

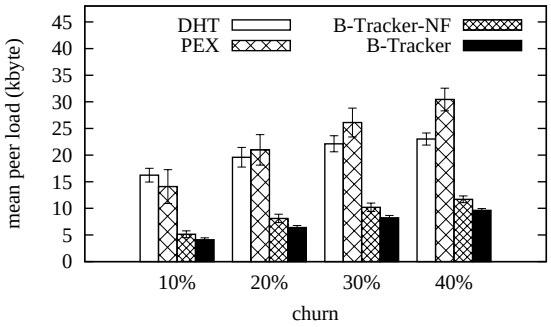

Figure 4.2: Efficiency, swarm size 250

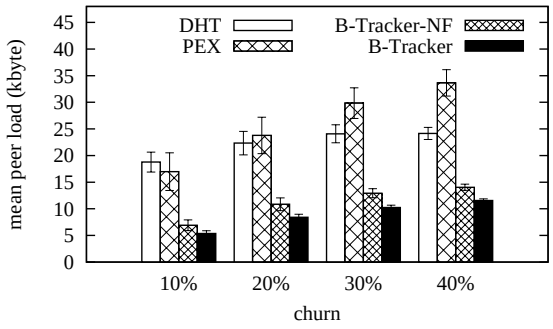

Figure 4.3: Efficiency, swarm size 450

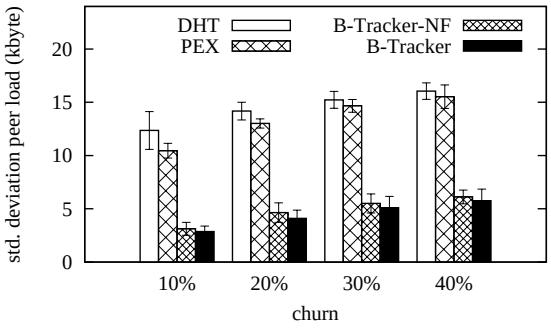

Figure 4.4: Load balancing, swarm size 50

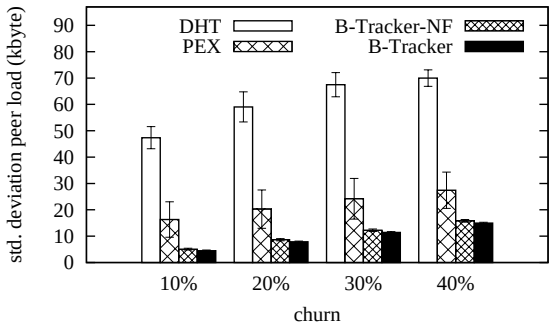

Figure 4.5: Load balancing, swarm size 250

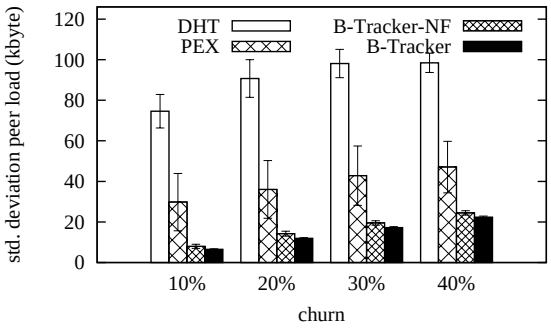

Figure 4.6: Load balancing, swarm size 450

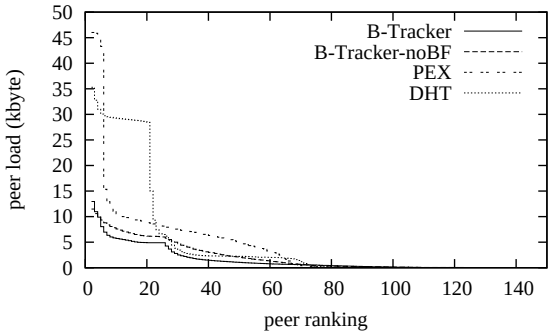

Figure 4.7: Peer load, swarm size 50, 10% churn

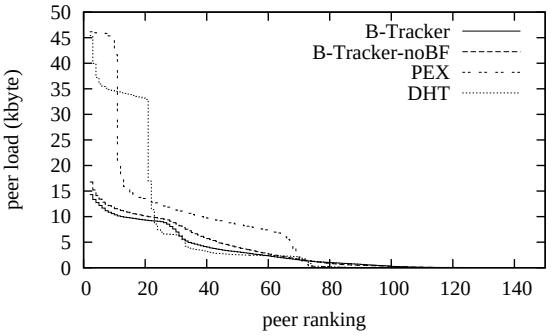

Figure 4.8: Peer load, swarm size 50, 20% churn

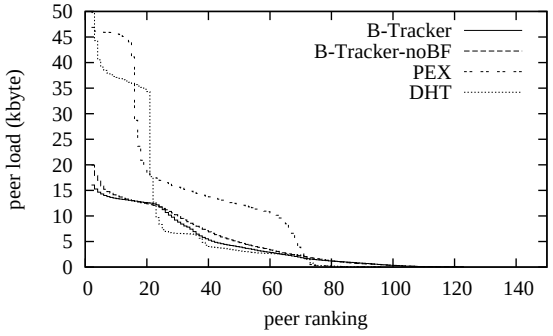

Figure 4.9: Peer load, swarm size 50, 30% churn

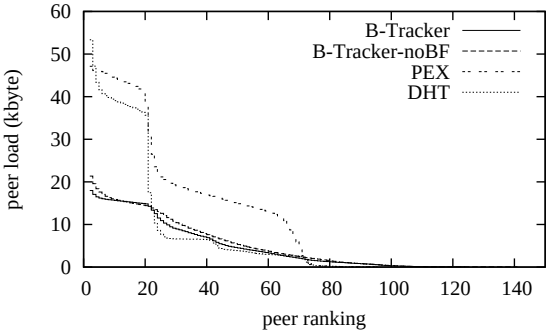

Figure 4.10: Peer load, swarm size 50, 40% churn

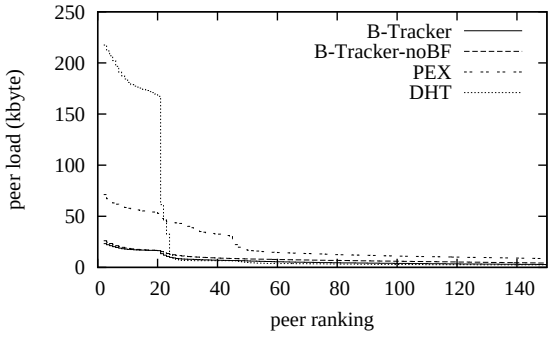

Figure 4.11: Peer load, swarm size 250, 10% churn

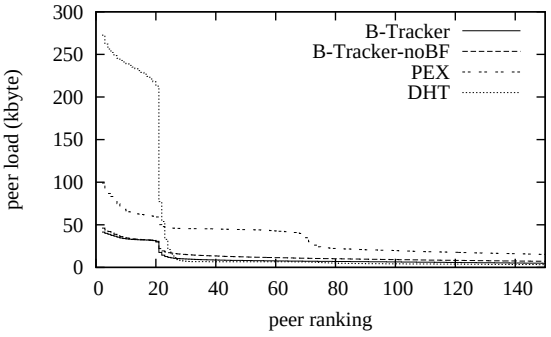

Figure 4.12: Peer load, swarm size 250, 20% churn

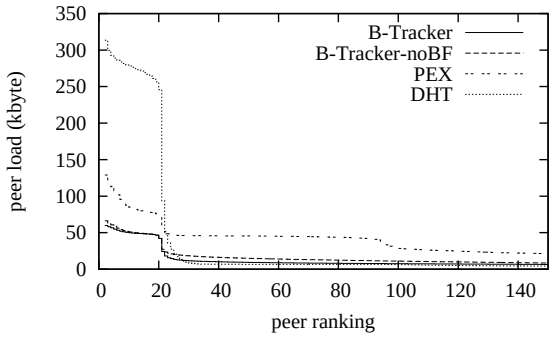

Figure 4.13: Peer load, swarm size 250, 30% churn

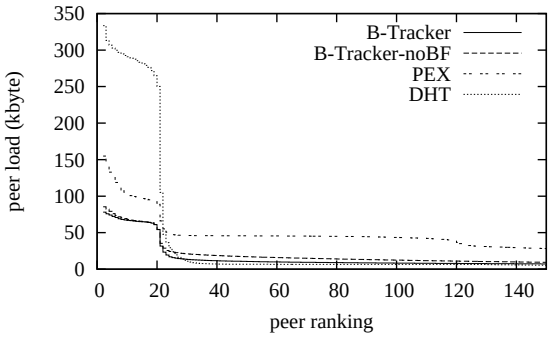

Figure 4.14: Peer load, swarm size 250, 40% churn

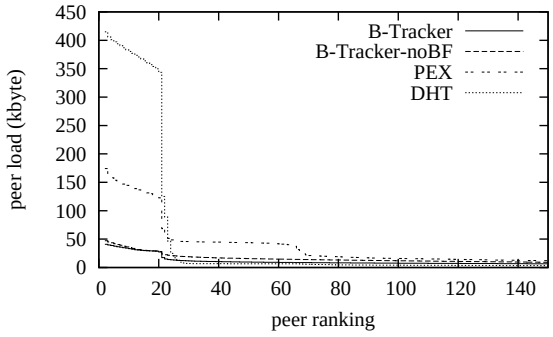

Figure 4.15: Peer load, swarm size 450, 10% churn

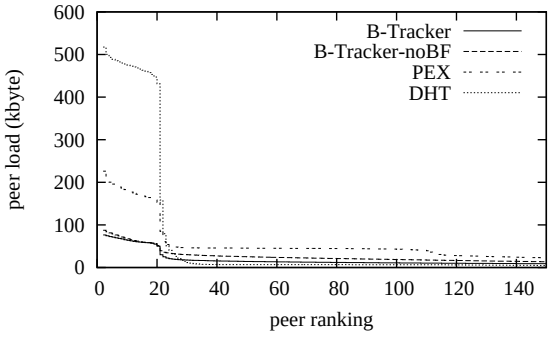

Figure 4.16: Peer load, swarm size 450, 20% churn

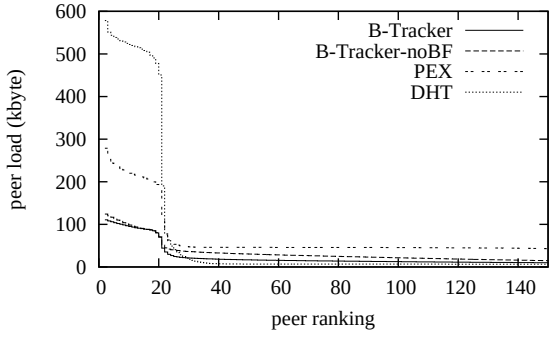

Figure 4.17: Peer load, swarm size 450, 30% churn

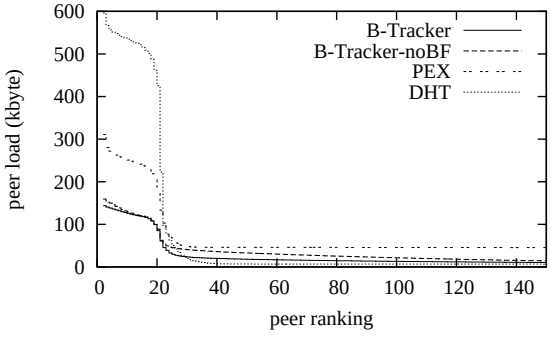

Figure 4.18: Peer load, swarm size 450, 40% churn

Chapter 5

Conclusions and Future Work

This paper introduced B-Tracker, a pull-based fully-distributed P2P tracker. B-Tracker improves load balancing by increasing the number of replicas proportionally to content popularity, improving load balancing. Bloom filters are used to improve efficiency, eliminating irrelevant providers from tracker replies, which can be achieved due to its pull approach. The pull approach also eliminates providers being sent to peers which are not interested in receiving new providers.

Simulations show that B-Tracker achieves a better load-balancing effects and higher efficiency than other distributed trackers. A pure DHT approach shows poor load balancing because it uses a fixed replication factor. PEX improves load balancing of the DHT approach but loses efficiency, since peers exchange messages which may not be of interest. Finally, a larger swarm size and higher churn produce only small degradation in B-Tracker's both load balancing and efficiency. The use of Bloom filters as suggested helps a further increase in system efficiency.

Future work includes analyzing security aspects with malicious peers and trackers, establishing theoretical bounds for B-Tracker operations, investigating in locality-awareness [9] of secondary trackers, and deploying B-Tracker on a real P2P system, to measure and analyze with additional swarm sizes and churn rates.

Acknowledgment

This work has been performed partially in the framework of the EU ICT STREP SmoothIT (FP7-2008-ICT-216259).

Bibliography

[1] BitTorrentPeerExchangeConventions - TheoryOrg. `http://wiki.theory.org/BitTorrentPeerExchangeConventions`, last visited: April 2011.

[2] Distributed Tracker - Tribler. `http://www.tribler.org/trac/wiki/DistributedTracker`, last visited: March 2011.

[3] Peer Exchange. `http://wiki.vuze.com/w/Peer_Exchange`, last visited: March 2011.

[4] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] B. Cohen. Incentives Build Robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, Berkeley, CA, USA, June 2003.

[6] S. A. Crosby and D. S. Wallach. An Analysis of BitTorrent's Two Kademlia-Based DHTs. Technical Report TR-07-04, Department of Computer Science, Rice University, June 2007.

[7] A. A. Hamra, A. Legout, and C. Barakat. Understanding the Properties of the BitTorrent Overlay. *CoRR*, abs/0707.1820, 2007.

[8] F. Hecht, T. Bocek, and D. Hausheer. The Pirate Bay 2008-12 Dataset. `http://www.csg.uzh.ch/publications/data/piratebay/`, December 2008.

[9] F. Lehrieder, S. Oechsner, T. Hoßfeld, Z. Despotovic, W. Kellerer, and M. Michel. Can P2P-Users Benefit from Locality-Awareness? In *Peer-to-Peer Computing*, pages 1–9, 2010.

[10] V. G. Papanicolaou, G. E. Kokolakis, and S. Boneh. Asymptotics for the random coupon collector problem. *Journal of Computational and Applied Mathematics*, 93(2):95 – 105, 1998.

[11] M. Steiner, T. En-Najjary, and E. W. Biersack. A Global View of KAD. In *7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 117–122, New York, NY, USA, 2007. ACM.

[12] TomP2P Project Site. `http://tomp2p.net`, last visited: April 2011.